

# Overview of Parallel I/O on HPC Systems

John Shalf  
NERSC  
SDSA



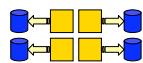
## Parallel I/O

- Requirements
  - Write data from multiple processors into a single file
  - File can be read in the same manner regardless of the number of CPUs that read from or write to the file. (e.g. we want to see the logical data layout... not the physical layout)
  - Do so with the same performance as writing one-file-per-processor (the uber-goal!!!)
- Scientific Data Schemas
  - 3D block structured grids
  - Particle data
  - Unstructured Cell Data (FEM codes)
  - AMR Data
  - Lots of other stuff, but not discussed here ...
- File Formats
  - ASCII
  - Raw Binary (F77 vs. C/C++)
  - Domain-specific/Community File Formats: FITS, PDB, Plot3D, CASE
  - Self-describing portable binary file formats: Silo, NetCDF, HDF.

## Parallel I/O for Block Structured Grids

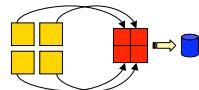
### One File Per Process

- Terrible for HPSS!
- Difficult to manage



### Parallel I/O into a single file

- Raw MPI-IO
- pHDF5 pNetCDF

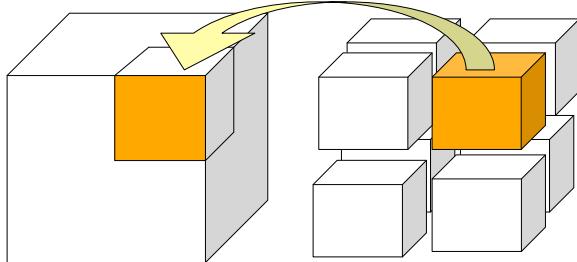


### Chunking into a single file

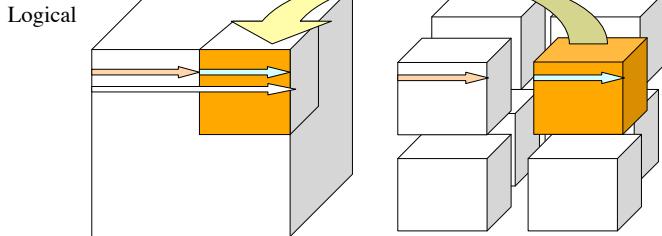
- Saves cost of reorganizing data
- Depend on API to hide physical layout
- (e.g. expose user to logically contiguous array even though it is stored physically as domain-decomposed chunks)



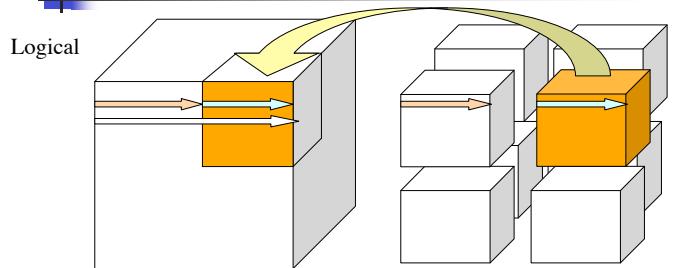
## 3D (reversing the domain decomp)



## 3D (reversing the decomps)



## 3D (block alignment issues)



Physical

720 bytes	720 bytes	8192 bytes	
-----------	-----------	------------	--

- Block updates require mutual exclusion
- Block thrashing on distributed PS
- I/O efficiency for sparse updates! (8k block required for 720 byte I/O operation)
- Unaligned block accesses can kill performance! (but are necessary in practical I/O solutions)



## Accelerator Modeling Data

- Point data
  - Electrons or protons
  - Millions or billions in a simulation
  - Distribution is non-uniform
    - . Fixed distribution at start of simulation
    - . Change distribution (load balancing) each iteration
- Attributes of a point
  - Location: (double) x,y,z
  - Phase: (double) mx,my,mz
  - ID: (int64) id
  - Other attributes



## Accelerator Modeling Data

### Storage Format

	P1	P2	P3	...	Xn
X	X1 X2 X3 X4 X5 X6 ..				
Y	Y1 Y2				Yn
Z					
...					
	2k particles	380 p	1k particles		



## Accelerator Modeling Benchmark

Mode	Global Performance	Per Processor Performance
One file per processor Raw binary	1288 MB/s	20 MB/s
Parallel I/O (1-file) Raw binary MPI-IO	241 MB/s	3 MB/s
Parallel I/O (1-file) pHDF5 -- H5Part	773 MB/s	12 MB/s

Seaborg: 64nodes,  
1024 processors,  
780 Gbytes of data total



## Accelerator Modeling Data

### Storage Format

X	0   X1 X2 X3 X4 X5 X6 X7   ...   Xn	NX-I
Y	Y1 Y2	NX + NY -I
Z	NY	
...		

Laid out sequentially on disk

Some formats are interleaved,

but causes problems for data analysis

Easier to reorganize in memory than on disk!



## Accelerator Modeling Data

### Calculate Offsets using Collective (AllGather)

Then write to mutually exclusive sections of array

X	P1	P2	P3	One array at a time
X	2k elements	380 elem	1k elements	→
Y				→
Z				→
...				→
	2k particles	380 p	1k particles	

Still suffers from alignment issues...



## Other Data Storage Types?

### Unstructured Cell Data

- 1D array of cell types
- 1D array of vertices (*x,y,z locations*)
- 1D array of cell connectivity
- Domain decomposition has similarity with particles, but must handle ghost cells

### AMR Data

- Chombo: Each 3D AMR grid occupies distinct section of 1D array on disk (*one array per AMR level*).
- Enzo (Mike Norman, UCSD): One file per processor (*each file contains multiple grids*)
- BoxLib: One file per grid (*each grid in the AMR hierarchy is stored in a separate, cleverly named, file*)



## Common Themes?

- Two patterns for parallel I/O into single file
  - >1D I/O: Each processor writes in a strided access pattern simultaneously to disk (*can be better organized... eg. PANDA*)
  - 1D I/O: Each processor writes to distinct subsections of 1D array (*or more than one array*)
- Three Storage Strategies
  - One file per processor (*terrible for HPSS!!!*)
  - One file per program: reverse domain decomp
  - One file per program: chunked output



## Storage Formats

- ASCII
  - Slow
  - Takes more space!
  - Inaccurate
- Binary
  - Nonportable (eg. byte ordering and types sizes)
  - Not future proof
  - Parallel I/O using MPI-IO
- Self-Describing formats
  - NetCDF/HDF4
  - HDF5

Parallel I/O using: pHDF5/pNetCDF (hides MPI-IO)

## Suggests 2 Benchmarks

- 2D-3D I/O patterns (striding)
  - 1 file per processor (*Raw Binary and HDF5*)
    - . Raw binary assesses peak performance
    - . HDF5 determines overhead of metadata, data encoding, and small accesses associated with storage of indices and metadata
  - 1-file reverse domain decomp (*Raw MPI-IO and pHDF5*)
    - . MPI-IO is baseline (peak performance)
    - . Assess pHDF5 or pNetCDF implementation overhead
  - 1-file chunked (*Raw MPI-IO and pHDF5*)
- 1D I/O patterns (writing to distinct 1D offsets)
  - Same as above, but for 1D data layouts
  - 1-file per processor is same in both cases
- *MadBench?*



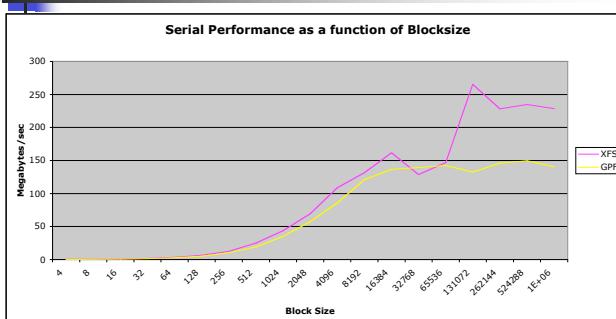
## GPFS MPI-I/O Experiences

nTasks	I/O Rate 16 Tasks/node	I/O Rate 8 tasks per node
8	-	131 Mbytes/sec
16	7 Mbytes/sec	139 Mbytes/sec
32	11 Mbytes/sec	217 Mbytes/sec
64	11 Mbytes/sec	318 Mbytes/sec
128	25 Mbytes/sec	471 Mbytes/sec

- Block domain decomp of  $512^3$  3D 8-byte/element array in memory written to disk as single undecomposed  $512^3$  logical array.
- Average throughput for 5 minutes of writes x 3 trials
- Issue is related to LAPI lock contention...



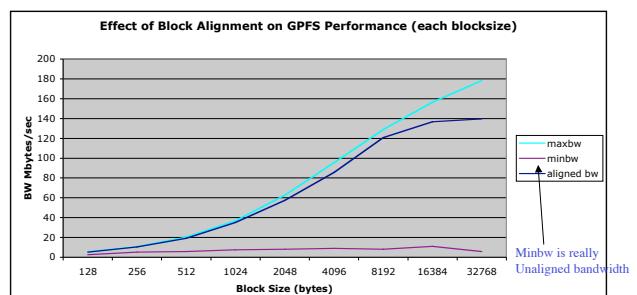
## GPFS: BW as function of write length



Block Aligned on disk!  
Page Aligned in memory!

Amdahl's law effects for  
Metadata storage...

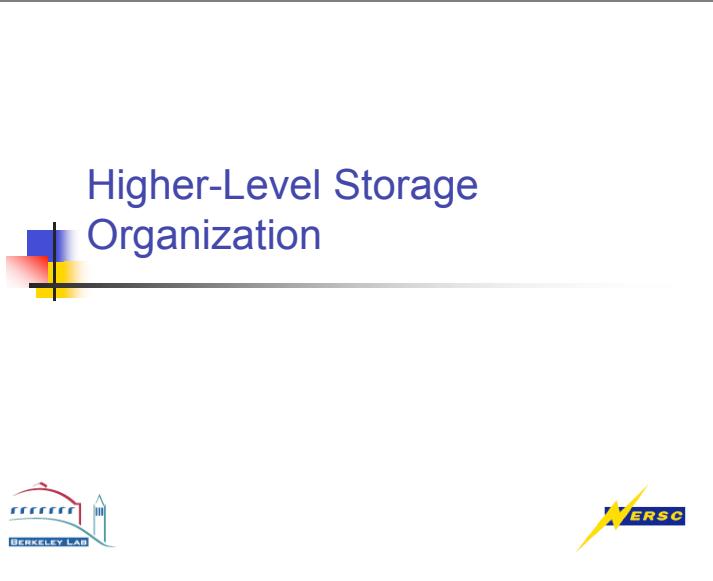
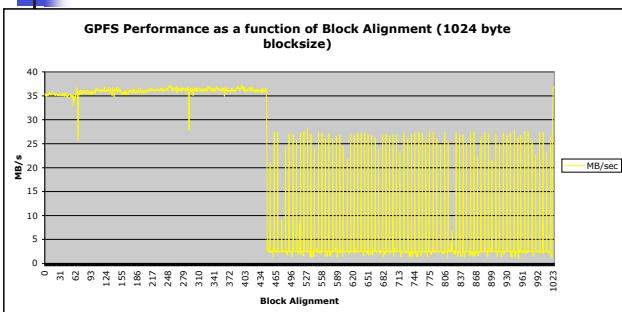
## GPFS (unaligned accesses)



Unaligned access sucks!



## GPFS: Unaligned accesses



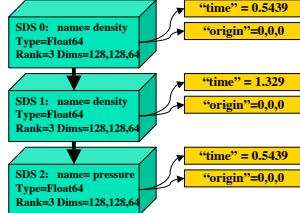
## HDF4/NetCDF Data Model

- Datasets

  - Name
  - Datatype
  - Rank,Dims

- Attributes

  - Key/value pair
  - DataType and length



## HDF4/NetCDF Data Model

- Datasets

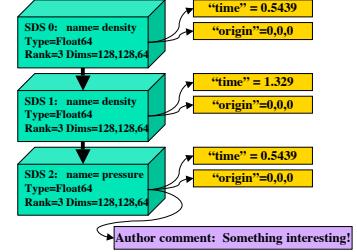
  - Name
  - Datatype
  - Rank,Dims

- Attributes

  - Key/value pair
  - DataType and length

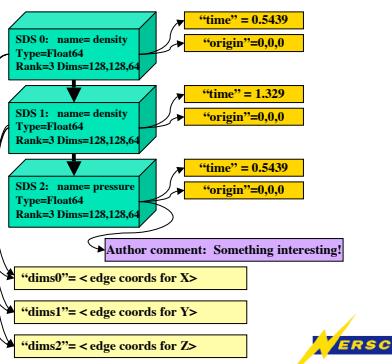
- Annotations

  - Freeform text
  - String Termination



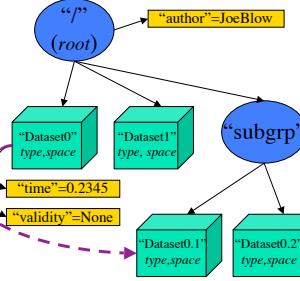
## HDF4/NetCDF Data Model

- Datasets**
  - Name
  - Datatype
  - Rank,Dims
- Attributes**
  - Key/value pair
  - DataType and length
- Annotations**
  - Freeform text
  - String Termination
- Dimensions**
  - Edge coordinates
  - Shared attribute



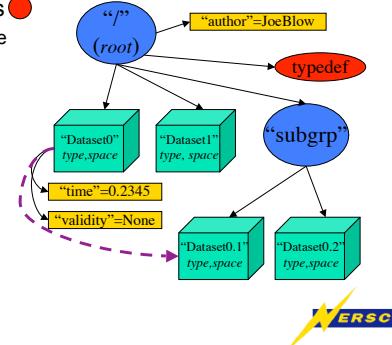
## HDF5 Data Model

- Groups**
  - Arranged in directory hierarchy
  - root group is always '/'
- Datasets**
  - Dataspace
  - Datatype
- Attributes**
  - Bind to Group & Dataset
- References**
  - Similar to softlinks
  - Can also be subsets of data



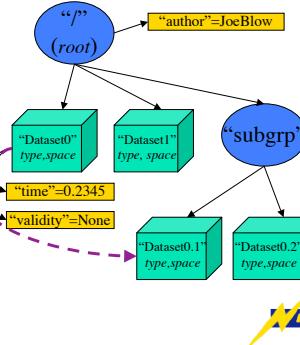
## HDF5 Data Model (funky stuff)

- Complex Type Definitions**
  - Not commonly used feature of the data model.
  - Potential pitfall if you commit complex datatypes to your file
- Comments**
  - Yes, annotations actually do live on.



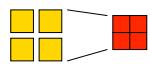
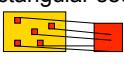
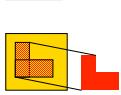
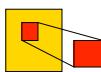
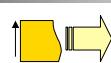
## HDF5 Data Model (caveats)

- Flexible/Simple Data Model**
  - You can do anything you want with it!
  - You typically define a higher level data model on top of HDF5 to describe domain-specific data relationships
  - Trivial to represent as XML!
- The perils of flexibility!**
  - Must develop community agreement on these data models to share data effectively
  - Multi-Community-standard data models required across for reusable visualization tools
  - Preliminary work on Images and tables



## Data Storage Layout / Selections

- Elastic Arrays**
- Hyperslabs**
  - Logically contiguous chunks of data
  - Multidimensional Subvolumes
  - Subsampling (striding, blocking)
- Union of Hyperslabs**
  - Reading a non-rectangular sections
- Gather/Scatter**
- Chunking**
  - Usually for efficient Parallel I/O

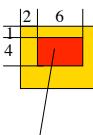


## Dataspace Selections (H5S)

Transfer a subset of data from disk to fill a memory buffer

### Disk Dataspace

```
H5Sselect_hyperslab(disk_space, H5S_SELECT_SET,
offset[3]={1,2},NULL,count[2]={4,6},NULL)
```



### Memory Dataspace

```
mem_space = H5S_ALL
```

Or

```
mem_space = H5Dcreate(rank=2,dims[2]={4,6});
```



### Transfer/Read operation

```
H5Dread(dataset,mem_datatype, mem_space, disk_space,
H5P_DEFAULT, mem_buffer);
```



## Dataspace Selections (H5S)

Transfer a subset of data from disk to subset in memory

### Disk Dataspace

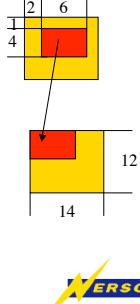
```
H5Sselect_hyperslab(disk_space, H5S_SELECT_SET,
    offset[3]={1,2},NULL,count[2]={4,6},NULL)
```

### Memory Dataspace

```
mem_space = H5Dcreate_simple(rank=2,dims[2]={12,14});
H5Sselect_hyperslab(mem_space, H5S_SELECT_SET,
    offset[3]={0,0},NULL,count[2]={4,6},NULL)
```

### Transfer/Read operation

```
H5Dread(dataset,mem_datatype, mem_space, disk_space,
    H5P_DEFAULT, mem_buffer);
```

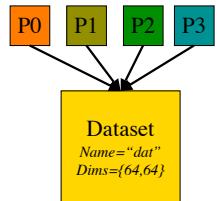


## pHDF5 (example 1)

- File open requires explicit selection of Parallel I/O layer.
- All PE's collectively open file and declare the overall size of the dataset.

### All MPI Procs!

```
props = H5Pcreate(H5P_FILE_ACCESS);
/* Create file property list and set for Parallel I/O */
H5Pset_fapl_mpio(prop, MPI_COMM_WORLD,
    MPI_INFO_NULL);
file=H5Fcreate(filename,H5F_ACC_TRUNC,
    H5P_DEFAULT,props); /* create file */
H5Pclose(props); /* release the file properties list */
filespace = H5Screate_simple(rank=2,dims[2]={64,64},
    NULL)
dataset = H5Dcreate(file,"dat",H5T_NATIVE_INT,
    space,H5P_DEFAULT); /* declare dataset */
```

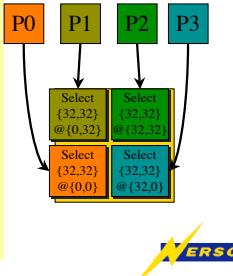


## pHDF5 (example 1 cont...)

- Each proc selects a hyperslab of the dataset that represents its portion of the domain-decomposed dataset and read/write collectively or independently.

### All MPI Procs!

```
/* select portion of file to write to */
H5Sselect_hyperslab(filespace, H5S_SELECT_SET,
    start= P0{0,0}:P1{0,32}:P2{32,32}:P3{32,0},
    stride= {32,1},count={32,32},NULL);
/* each proc independently creates its memspace */
memspace = H5Screate_simple(rank=2,dims={32,32},
    NULL);
/* setup collective I/O prop list */
xfer_plist = H5Pcreate (H5P_DATASET_XFER);
H5Pset_dxpl_mpio(xfer_plist, H5FD_MPIO_COLLECTIVE);
H5Dwrite(dataset,H5T_NATIVE_INT, memspace, filespace,
    xfer_plist, local_data); /* write collectively */
```



## Serial I/O Benchmarks

System	HDF4.1r5 (netCDF)	HDF5 v1.4.4	FlexIO (Custom)	F77 Unf
SGI Origin 3400 (escher.nersc.gov)	111M/s	189M/s	180M/s	140M/s
IBM SP2 (seaborg.nersc.gov)	65M/s	127M/s	110M/s	110M/s
Linux IA32 (platinum.ncsa.uiuc.edu)	34M/s	40M/s	62M/s	47M/s
Linux IA64 Teragrid node (titan.ncsa.uiuc.edu)	26M/s	83M/s	77M/s	112M/s
NEC/Cray SX-6 (rime.cray.com)				

- Write 5-40 datasets of 128^3 DP float data
- Single CPU (multiple CPU's can improve perf. until interface saturates)

Average of 5 trials



## GPFS MPI-I/O Experiences

nTasks	I/O Rate 16 Tasks/node	I/O Rate 8 tasks per node
8	-	131 Mbytes/sec
16	7 Mbytes/sec	139 Mbytes/sec
32	11 Mbytes/sec	217 Mbytes/sec
64	11 Mbytes/sec	318 Mbytes/sec
128	25 Mbytes/sec	471 Mbytes/sec

- Block domain decomp of 512^3 3D 8-byte/element array in memory written to disk as single un-decomposed 512^3 logical array.
- Average throughput for 5 minutes of writes x 3 trials

